A Trainable Optimal Transport Embedding for Feature Aggregation

Grégoire Mialon*
Inria[†‡]
gregoire.mialon@inria.fr

Dexiong Chen*
Inria[†]
dexiong.chen@inria.fr

Alexandre d'Aspremont
CNRS - ENS[‡]
aspremon@ens.fr

Julien Mairal
Inria[†]
julien.mairal@inria.fr

Abstract

We address the problem of learning on large sets of features, motivated by the need of performing pooling operations in long biological sequences of varying sizes, with long-range dependencies, and possibly few labeled data. To address this challenging task, we introduce a parametrized embedding that aggregates the features from a given set according to the optimal transport plan between the set and a trainable reference. Our embedding (i) integrates an inductive bias akin to that of self-attention in transformers, *i.e* aggregating features based on similarity, while (ii) its kernel grounding provides a simple unsupervised learning mechanism. Our approach also allows end-to-end training of the reference. We demonstrate the effectiveness of our approach on biological sequences, achieving state-of-the-art results for protein fold recognition tasks. Our code is freely available at `https://github.com/claying/OTK`. A longer version of the paper is available at `https://arxiv.org/abs/2006.12065`.

1 Introduction

Many scientific fields such as bioinformatics require processing sets of features with positional information. These objects are delicate to manipulate due to varying lengths and potentially long-range dependencies between their elements. For many tasks, the difficulty is even greater since the sequences can be arbitrarily long, or only provided with few labels, or both.

Attention [2] and then transformers [27] were proposed to cope with such data. In particular, transformers led to major progress in many natural language processing (NLP) tasks [29] and to some extent in other fields relying on structured data such bioinformatics [22]. However, a major drawback of these models is their prohibitive number of parameters. In many tasks, it is thus very expensive or impossible to gather enough data to train such models. More recently, deep learning architectures specifically designed for sets have been proposed [16, 25]. Our experiments show that these approaches achieve mixed performance for long biological sequences of varying size.

To address these issues, we marry ideas from kernel methods [24] and optimal transport (OT) [20]: we embed feature vectors of a given set to a reproducing kernel Hilbert space (RKHS) and then perform a weighted pooling operation, with weights given by the transport plan between the set and a trainable reference. The motivation for using kernels is to provide a non-linear transformation of the input features before pooling, whereas OT allows to align the features on a trainable reference

*Equal contribution.

[†]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France.

[‡]D.I., UMR 8548, École Normale Supérieure, Paris, France.

1st NeurIPS workshop on Interpretable Inductive Biases and Physically Structured Learning (2020), virtual.

with fast algorithms [7]. Such combination provides us with a theoretically grounded embedding that can be learned either without any label, or with supervision. In the latter case, we will see that our model encodes prior "knowledge" in the sense that it is initialized with references that are representative of the data set. Our embedding can operate on large sets with varying size and model long-range dependencies when positional information is present. It shares the same inductive bias as transformers self-attention, *i.e*, aggregating features given a notion of similarity. We demonstrate its effectiveness on protein fold recognition tasks, among others, where we achieve state-of-the-art results. We also show promising results in natural language processing tasks.

2 Proposed Embedding

2.1 Optimal Transport for Sets of Feature Vectors

We handle sets of features in $\mathbb{R}^d$ and consider sets $\mathbf{x}$ living in

$$\mathcal{X} = \left\{\mathbf{x} \mid \mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \text{ such that } \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d \text{ for some } n \geq 1\right\}.$$

Elements of $\mathcal{X}$ are typically vector representations of local data structures, such as k-mers for sequences or words for sentences. The size of $\mathbf{x}$ denoted by n may vary, which is not an issue since the methods we introduce may take a sequence of any size as input and provide a fixed-size embedding.

Our pooling mechanism will be based on the transport plan between $\mathbf{x}$ and $\mathbf{x}'$ seen as weighted point clouds, which is a by-product of the optimal transport problem. OT has indeed been widely used in alignment problems [11]. Let $\mathbf{a}$ in Δ^n (probability simplex) and $\mathbf{b}$ in $\Delta^{n'}$ be the weights of the discrete measures $\sum_i \mathbf{a}_i \delta_{\mathbf{x}_i}$ and $\sum_j \mathbf{b}_j \delta_{\mathbf{x}'_j}$ with respective locations $\mathbf{x}$ and $\mathbf{x}'$, where $\delta_{\mathbf{x}}$ is the Dirac at position $\mathbf{x}$. Let $\mathbf{C}$ in $\mathbb{R}^{n \times n'}$ be a matrix representing the pairwise costs for aligning the elements of $\mathbf{x}$ and $\mathbf{x}'$. The entropic regularized Kantorovich relaxation of OT [20] from $\mathbf{x}$ to $\mathbf{x}'$ is

$$\min_{\mathbf{P} \in U(\mathbf{a}, \mathbf{b})} \sum_{ij} \mathbf{C}_{ij} \mathbf{P}_{ij} - \varepsilon H(\mathbf{P}), \tag{1}$$

where $H(\mathbf{P}) = -\sum_{ij} \mathbf{P}_{ij}(\log(\mathbf{P}_{ij}) - 1)$ is the entropic regularization with parameter ε, which controls the sparsity of $\mathbf{P}$, and U is the space of admissible couplings between $\mathbf{a}$ and $\mathbf{b}$:

$$U(\mathbf{a}, \mathbf{b}) = \{\mathbf{P} \in \mathbb{R}_+^{n \times n'} : \mathbf{P}\mathbf{1}_n = \mathbf{a} \text{ and } \mathbf{P}^\top \mathbf{1}_{n'} = \mathbf{b}\}.$$

In practice, $\mathbf{a}$ and $\mathbf{b}$ are uniform measures since we consider the mass to be evenly distributed between the points. $\mathbf{P}$ is called the transport plan, which carries the information on how to distribute the mass of $\mathbf{x}$ in $\mathbf{x}'$ with minimal cost.

2.2 Optimal Transport Embedding and Associated Kernel

We now present our embedding, starting with an infinite-dimensional variant living in a RKHS.

Infinite-dimensional embedding in RKHS. Kernel methods [24] map data living in a space $\mathcal{X}$ to a reproducing kernel Hilbert space $\mathcal{H}$, associated to a positive definite kernel K through a mapping function $\varphi : \mathcal{X} \to \mathcal{H}$, such that $K(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle_{\mathcal{H}}$. Given a set $\mathbf{x}$ and a (learned) reference $\mathbf{z}$ in $\mathcal{X}$ with p elements, we consider an embedding $\Phi_{\mathbf{z}}(\mathbf{x})$ which performs the following operations: (i) initial embedding of the elements of $\mathbf{x}$ and $\mathbf{z}$ to a RKHS $\mathcal{H}$; (ii) alignment of the elements of $\mathbf{x}$ to the elements of $\mathbf{z}$ via optimal transport; (iii) weighted linear pooling of the elements $\mathbf{x}$ into p bins, producing an embedding $\Phi_{\mathbf{z}}(\mathbf{x})$ in $\mathcal{H}^p$, which is illustrated in Figure 1. Before introducing more formal details, we note that our embedding relies on two main ideas:

- *Global similarity-based pooling using references.* Learning on large sets with long-range interactions may benefit from pooling to reduce the number of feature vectors. Our pooling rule follows an inductive bias akin to that of self-attention: elements that are relevant to each other for the task at hand should be pooled together. To this end, each element in the reference set corresponds to a pooling cell, where the elements of the input set are aggregated through a weighted sum. The weights simply reflect the similarity between the vectors of the input set and the current vector in the reference. Importantly, using a reference set enables to reduce the size of the "attention matrix" from quadratic to linear in the length of the input sequence.

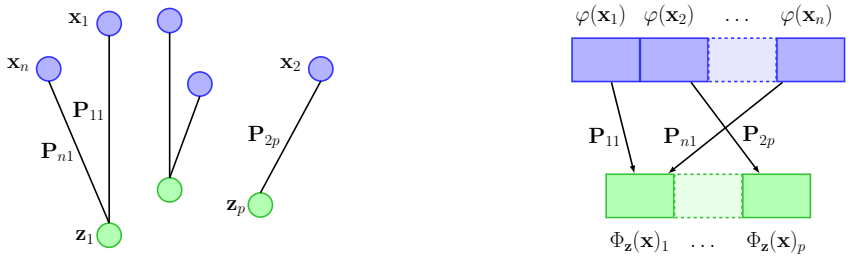

Figure 1: The input point cloud $\mathbf{x}$ is transported onto the reference $\mathbf{z} = (\mathbf{z}_1, \dots, \mathbf{z}_p)$ (left), yielding the optimal transport plan $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$ used to aggregate the embedded features and form $\Phi_{\mathbf{z}}(\mathbf{x})$ (right).

- *Computing similarity weights via OT.* A popular similarity score between two elements is their dot-product [27]. In this paper, we rather consider that elements of the input set should be pooled together if they align with the same part of the reference. Alignment scores can efficiently be obtained by computing the transport plan between the input and the reference sets: Sinkhorn's algorithm indeed enjoys fast solvers [7]. We are now in shape to give a formal definition.

Definition 2.1 (The optimal transport kernel embedding). Let $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ in $\mathcal{X}$ be an input set of feature vectors and $\mathbf{z} = (\mathbf{z}_1, \dots, \mathbf{z}_p)$ in $\mathcal{X}$ be a reference set with p elements. Let κ be a positive definite kernel, *e.g.*, Gaussian kernel, with RKHS $\mathcal{H}$ and $\varphi : \mathbb{R}^d \to \mathcal{H}$, its associated kernel embedding. Let $\boldsymbol{\kappa}$ be the $n \times p$ matrix which carries the comparisons $\kappa(\mathbf{x}_i, \mathbf{z}_j)$, before alignment. Then, the transport plan between $\mathbf{x}$ and $\mathbf{z}$, denoted by the $n \times p$ matrix $\mathbf{P}(\mathbf{x}, \mathbf{z})$, is defined as the unique solution of (1) when choosing the cost $\mathbf{C} = -\boldsymbol{\kappa}$, and our embedding is defined as

$$\Phi_{\mathbf{z}}(\mathbf{x}) := \sqrt{p} \times \left(\sum_{i=1}^{n} \mathbf{P}(\mathbf{x}, \mathbf{z})_{i1} \varphi(\mathbf{x}_i), \ \dots \ , \sum_{i=1}^{n} \mathbf{P}(\mathbf{x}, \mathbf{z})_{ip} \varphi(\mathbf{x}_i) \right) = \sqrt{p} \times \mathbf{P}(\mathbf{x}, \mathbf{z})^\top \varphi(\mathbf{x}),$$

where $\varphi(\mathbf{x}) := [\varphi(\mathbf{x}_1), \dots, \varphi(\mathbf{x}_n)]^\top$.

Interestingly, it is easy to show that our embedding $\Phi_{\mathbf{z}}(\mathbf{x})$ is associated to the positive definite kernel

$$K_{\mathbf{z}}(\mathbf{x}, \mathbf{x}') := \sum_{i,i'=1}^{n} \mathbf{P}_{\mathbf{z}}(\mathbf{x}, \mathbf{x}')_{ii'} \kappa(\mathbf{x}_i, \mathbf{x}'_{i'}) = \langle \Phi_{\mathbf{z}}(\mathbf{x}), \Phi_{\mathbf{z}}(\mathbf{x}') \rangle, \tag{2}$$

with $\mathbf{P}_{\mathbf{z}}(\mathbf{x}, \mathbf{x}') := p \times \mathbf{P}(\mathbf{x}, \mathbf{z}) \mathbf{P}(\mathbf{x}', \mathbf{z})^\top$. This is a weighted match kernel, with weights given by optimal transport in $\mathcal{H}$. $K_{\mathbf{z}}$ is related to a classical kernel, see Appendix B.1. Our embedding can be extended to use multiple references, similar to the notion of multi-head attention, and integrate positional information, see Appendix B.3. We now expose the benefits of this kernel formulation.

From infinite-dimensional kernel embedding to finite dimension. In some cases, $\varphi(\mathbf{x})$ is already finite-dimensional, which allows to compute the embedding $\Phi_{\mathbf{z}}(\mathbf{x})$ explicitly. This is particularly useful when dealing with large-scale data, as it enables us to use our method for supervised learning tasks without computing the Gram matrix, which grows quadratically in size with the number of samples. When φ is infinite or high-dimensional, it is nevertheless possible to use an approximation based on the Nyström method [32], which provides an embedding $\psi : \mathbb{R}^d \to \mathbb{R}^k$ such that $\langle \psi(\mathbf{x}_i), \psi(\mathbf{x}'_j) \rangle_{\mathbb{R}^k} \approx \kappa(\mathbf{x}_i, \mathbf{x}'_j)$ (see Appendix A.2). The corresponding embedding admits an explicit form $\psi(\mathbf{x}_i) = \kappa(\mathbf{w}, \mathbf{w})^{-1/2} \kappa(\mathbf{w}, \mathbf{x}_i)$, where $\kappa(\mathbf{w}, \mathbf{w})$ is the $k \times k$ Gram matrix of κ computed on the set $\mathbf{w} = \{\mathbf{w}_1, \dots, \mathbf{w}_k\}$ of anchor points and $\kappa(\mathbf{w}, \mathbf{x}_i)$ is in $\mathbb{R}^k$. Anchor points can be defined as centroids obtained by K-means, see [33], or learned by back-propagation for a supervised task, see [17]. Using such an approximation within our framework can be simply achieved by (i) replacing κ by a linear kernel and (ii) replacing each element $\mathbf{x}_i$ by its embedding $\psi(\mathbf{x}_i)$ in $\mathbb{R}^k$ and considering a reference set with elements in $\mathbb{R}^k$. By abuse of notation, we still use $\mathbf{z}$ for the new parametrization. The embedding, which we use in practice in all our experiments, becomes simply

$$\Phi_{\mathbf{z}}(\mathbf{x}) = \sqrt{p} \times \mathbf{P}(\psi(\mathbf{x}), \mathbf{z})^\top \psi(\mathbf{x}) \in \mathbb{R}^{p \times k}, \tag{3}$$

where p is the number of elements in $\mathbf{z}$. Next, we discuss how to learn the reference set $\mathbf{z}$.

Unsupervised learning of $\mathbf{z}$. In the fashion of the Nyström approximation, the p elements of $\mathbf{z}$ can be defined as the centroids obtained by K-means applied to all features from training sets in $\mathcal{X}$. The anchor points $\mathbf{w}$ and the references $\mathbf{z}$ may be then computed using similar algorithms; however, their mathematical interpretation differs as exposed above. The task of representing features (learning $\mathbf{w}$ in $\mathbb{R}^d$ for a specific κ) is decoupled from the task of aggregating (learning the reference $\mathbf{z}$ in $\mathbb{R}^k$).

Supervised learning of $\mathbf{z}$. $\mathbf{P}(\psi(\mathbf{x}), \mathbf{z})$ is computed using Sinkhorn's algorithm, [20]., recalled in Appendix A, which can be easily adapted to batches of samples $\mathbf{x}$, with possibly varying lengths, leading to GPU-friendly forward computations of $\Phi_{\mathbf{z}}$. All Sinkhorn's operations are differentiable, which enables $\mathbf{z}$ to be optimized with SGD through back-propagation, *e.g.*, for minimizing a classification or regression loss function when labels are available. In practice, a small number of Sinkhorn iterations (*e.g.*, 10) is sufficient to compute $\mathbf{P}(\psi(\mathbf{x}), \mathbf{z})$. Since the anchors $\mathbf{w}$ in the embedding layer below can also be learned end-to-end [17], our embedding can be used as a module injected into any model, *e.g*, a deep network. We typically initialize the model with parameters given by unsupervised training. Finally, the relationship between OTKE and self-attention is discussed in Appendix B.4.

3 Experiments

We show the effectiveness of the OTKE in an important protein fold classification task on the SCOP 1.75 data set, where samples can be expressed as large sets with potentially few labels. We evaluate our embedding alone in unsupervised or supervised settings. In unsupervised settings, we train a linear classifier on top of features provided by our unsupervised embedding, or an unsupervised baseline. In supervised settings, the same model is initialized with our unsupervised method and then trained end-to-end. The hyper-parameters we tuned include number of supports and references p, q, entropic regularization in OT ε, the bandwidth of Gaussian kernels and the regularization parameter of the linear classifier. The best values of ε and the bandwidth were found , while the regularization parameter needed to be cross-validated. Additional results, data sets and implementation details can be found in Appendix D: we show state-of-the-art results for another bio-informatics task, the detection of chromatin profiles, and promising results in natural language processing.

We follow the protocol described by [13]. The dataset contains $19,245$ sequences from $1,195$ different classes of fold, *i.e* less than 20 labels in average per class. The sequence lengths vary from tens to thousands. Each element of a sequence is a 45-dimensional vector. The objective is to classify the sequences to fold classes, which corresponds to a multiclass classification problem. The features fed to the linear classifier are the output of our embedding with φ the Gaussian kernel mapping on k-mers (subsequences of length k) with k fixed to 10, which is known to perform well in this task [4]. The number of anchor points for Nyström method is fixed to 1024 and 512 respectively for unsupervised and supervised setting. In the unsupervised setting, we compare our method to state-of-the-art unsupervised method for this task: CKN [4], which performs a global mean pooling in contrast to the global adaptive pooling performed by our embedding. In the supervised setting, we compare the same model to the following supervised models: CKN, Recurrent Kernel Networks (RKN) [5], a CNN with 10 convolutional layers named DeepSF [13], Rep the Set [25] and Set Transformer [16], using the public implementations by their authors. Rep the Set and Set Transformer are used on the top of a convolutional layer of the same filter size as CKN to extract k-mer features. Their model hyper-parameters, weight decay and learning rate are tuned in the same way as our models. The default architecture of Set Transformer did not perform well due to overfitting. We thus used a shallower architecture with one ISAB, one PMA and one linear layer, similar to the one-layer architectures of CKN and our model. The results are shown in Table 1.

Table 1: Classification accuracy (top 1/5/10) on test set for SCOP 1.75 for different unsupervised and supervised baselines, averaged from 10 different runs. (q references $\times$ p supports).

Method	Unsupervised	Supervised
DeepSF [13]	Not available.	73.0/90.3/94.5
CKN [4]	81.8±0.8/92.8±0.2/95.0±0.2	84.1±0.1/94.3±0.2/96.4±0.1
RKN [5]	Not available.	85.3±0.3/95.0±0.2/96.5±0.1
Set Transformer [16]	Not available.	79.2±4.6/91.5±1.4/94.3±0.6
Approximate Rep the Set [25]	Not available.	84.5±0.6/94.0±0.4/95.7±0.4
Ours (dot-product instead of OT)	78.2±1.9/93.1±0.7/96.0±0.4	87.5±0.3/95.5±0.2/96.9±0.1
Ours (Unsup.: 1×100 / Sup.: 5×10)	**85.8±0.2/95.3±0.1/96.8±0.1**	**88.7±0.3/95.9±0.2/97.3±0.1**

Our embedding outperforms all baselines. Surprisingly, our unsupervised model also achieves better results than most supervised baselines. In contrast, Set Transformer does not perform well, possibly because its implementation was not designed for sets with varying sizes, and tasks with few annotations. The optimal number of references and supports appear to be data dependent. Generally, each reference has a cardinality lower than the length of the input sets. Using multiple references improves performance in supervised settings only, see Appendix D.3. We observed our method to be much faster than RepSet, as fast as Set Transformer, yet slower than ApproxRepSet (D.3). Using the OT plan as similarity score yields better accuracies than the dot-product between the input sets and the references (see Table 1; 15; 16). A future direction would be exploring the content of the references to see whether it can be interpreted, and used to encode real prior knowledge.

Acknowledgments

JM, GM and DC were supported by the ERC grant number 714381 (SOLARIS project) and by ANR 3IA MIAI@Grenoble Alpes, (ANR-19-P3IA-0003). AA would like to acknowledge support from the *ML and Optimisation* joint research initiative with the *fonds AXA pour la recherche* and Kamet Ventures, a Google focused award, as well as funding by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute). DC and GM thank Laurent Jacob, Louis Martin, François-Pierre Paty and Thomas Wolf for useful discussions.

References

[1] J.-M. Andreoli. Convolution, attention and structure embedding. In *arXiv preprint arXiv: 1905.01289v5*, 2019.

[2] D. Bahdanau, K. Cho, and J. Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

[3] A. Buades, B. Coll, and J.-M. Morel. Non-Local Means Denoising. *Image Processing On Line*, 1:208–212, 2011.

[4] D. Chen, L. Jacob, and J. Mairal. Biological sequence modeling with convolutional kernel networks. *Bioinformatics*, pages 35(18):3294–3302, 2019.

[5] D. Chen, L. Jacob, and J. Mairal. Recurrent kernel networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[6] J.-B. Cordonnier, A. Loukas, and M. Jaggi. On the relationship between self-attention and convolutional layers. In *International Conference on Learning Representations (ICLR)*, 2020.

[7] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[8] M. Cuturi and A. Doucet. Fast computation of wasserstein barycenters. In *International Conference on Machine Learning (ICML)*, 2013.

[9] Z. Dai, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[11] E. Grave, A. Joulin, and Q. Berthet. Unsupervised alignment of embeddings with wasserstein procrustes. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

[12] N. Ho, X. Nguyen, M. Yurochkin, H. H. Bui, V. Huynh, and D. Phung. Multilevel clustering via wasserstein means. In *International Conference on Machine Learning (ICML)*, 2017.

[13] J. Hou, B. Adhikari, and J. Cheng. Deepsf: deep convolutional neural network for mapping-protein sequences to folds. *Bioinformatics*, pages 34(8):1295–1303, 2019.

[14] N. Kitaev, L. Kaiser, and A. Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations (ICLR)*, 2020.

[15] S. Kolouri, Y. Zou, and G. K. Rohde. Sliced wasserstein kernels for probabilit distributions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[16] J. Lee, Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation invariant neural networks. In *International Conference on Machine Learning (ICML)*, 2019.

[17] J. Mairal. End-to-end kernel learning with supervised convolutional kernel networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[18] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[19] P. Michel, O. Levy, and G. Neubig. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[20] G. Peyré and M. Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–206, 2019.

[21] A. Raganato, Y. Scherrer, and T. Jörg. Fixed encoder self-attention patterns in transformer-based machine translation. In *arXiv preprint arXiv: 2002.10260*, 2020.

[22] A. Rives, S. Goyal, J. Meier, D. Guo, M. Ott, C. L. Zitnick, J. Ma, and R. Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. In *bioRxiv 622803*, 2019.

[23] Y. Rubner, C. Tomasi, and L. J. Guibad. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40:99–121, 2000.

[24] B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[25] K. Skianis, G. Nikolentzos, S. Limnios, and M. Vazirgiannis. Rep the set: Neural networks for learning set representations. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[26] Y.-H. H. Tsai, S. Bai, M. Yamada, L.-P. Morency, and R. Salakhutdinov. Transformer dissection: A unified understanding of transformer's attention via the lens of kernel. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[28] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

[29] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: a multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations (ICLR)*, 2019.

[30] X. Wang, R. B. Girshick, A. Gupta, and K. He. Non-local neural networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[31] Y. Weiqiu, S. Sun, and M. Iyyer. Hard-coded gaussian attention for neural machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[32] C. K. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2001.

[33] K. Zhang, I. W. Tsang, and J. T. Kwok. Improved nyström low-rank approximation and error analysis. In *International Conference on Machine Learning (ICML)*, 2008.

[34] J. Zhou and O. G. Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature methods*, 12(10):931–934, 2015.

Appendix

Appendix A provides some background on notions used throughout the paper; Appendix B provides technical details that were omitted in the paper for readability; Appendix C contains the proofs skipped in the paper; Appendix D provides additional experimental results as well as details on our protocol for reproducibility.

A Additional Background

This section provides some background on attention and transformers, Sinkhorn's algorithm and the relationship between optimal transport based kernels and positive definite histogram kernels.

A.1 Sinkhorn's Algorithm: Fast Computation of $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$

Without loss of generality, we consider here κ the linear kernel. We recall that $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$ is the solution of an optimal transport problem, which can be efficiently solved by Sinkhorn's algorithm [20] involving matrix multiplications only. Specifically, Sinkhorn's algorithm is an iterative matrix scaling method that takes the opposite of the pairwise similarity matrix $\mathbf{K}$ with entry $\mathbf{K}_{ij} := \langle \mathbf{x}_i, \mathbf{z}_j \rangle$ as input $\mathbf{C}$ and outputs the optimal transport plan $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z}) = \text{Sinkhorn}(\mathbf{K}, \varepsilon)$. Each iteration step ℓ performs the following updates

$$\mathbf{u}^{(\ell+1)} = \frac{1/n}{\mathbf{E}\mathbf{v}^{(\ell)}} \quad \text{and} \quad \mathbf{v}^{(\ell+1)} = \frac{1/p}{\mathbf{E}^\top \mathbf{u}^{(\ell)}}, \tag{4}$$

where $\mathbf{E} = e^{\mathbf{K}/\varepsilon}$. Then the matrix $\text{diag}(\mathbf{u}^{(\ell)})\mathbf{E}\text{diag}(\mathbf{v}^{(\ell)})$ converges to $\mathbf{P}_\kappa(\mathbf{x}, \mathbf{z})$ when ℓ tends to ∞. However when ε becomes too small, some of the elements of a matrix product $\mathbf{E}\mathbf{v}$ or $\mathbf{E}^\top \mathbf{u}$ become null and result in a division by 0. To overcome this numerical stability issue, computing the multipliers $\mathbf{u}$ and $\mathbf{v}$ is preferred (see *e.g.* [20, Remark 4.23]). This algorithm can be easily adapted to a batch of data points $\mathbf{x}$, and with possibly varying lengths via a mask vector masking on the padding positions of each data point $\mathbf{x}$, leading to GPU-friendly computation. More importantly, all the operations above at each step are differentiable, which enables $\mathbf{z}$ to be optimized through back-propagation. Consequently, this module can be injected into any deep networks.

A.2 Nyström Method

The Nyström method [32], provides an embedding $\psi : \mathbb{R}^d \to \mathbb{R}^k$ such that

$$\langle \psi(\mathbf{x}_i), \psi(\mathbf{x}'_j) \rangle_{\mathbb{R}^k} \approx \kappa(\mathbf{x}_i, \mathbf{x}'_j).$$

Concretely, the Nyström method consists in projecting points from the RKHS $\mathcal{H}$ onto a linear subspace $\mathcal{F}$, which is parametrized by k anchor points $\mathcal{F} = \text{Span}(\varphi(\mathbf{w}_1), \ldots, \varphi(\mathbf{w}_k))$. The corresponding embedding admits an explicit form $\psi(\mathbf{x}_i) = \kappa(\mathbf{w}, \mathbf{w})^{-1/2}\kappa(\mathbf{w}, \mathbf{x}_i)$, where $\kappa(\mathbf{w}, \mathbf{w})$ is the $k \times k$ Gram matrix of κ computed on the set $\mathbf{w} = \{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ of anchor points and $\kappa(\mathbf{w}, \mathbf{x}_i)$ is in $\mathbb{R}^k$. Then, there are several ways to learn the anchor points: (a) they can be chosen as random points from data; (b) they can be defined as centroids obtained by K-means, see [33]; (c) they can be learned by back-propagation for a supervised task, see [17].

A.3 Attention and Transformers

We clarify the concept of attention — a mechanism yielding a context-dependent embedding for each element of $\mathbf{x}$ — as a special case of non-local operations [3, 30], so that it is easier to understand its relationship to the OTK. Let us assume we are given a set $\mathbf{x} \in \mathcal{X}$ of length n. A non-local operation on $\mathbf{x}$ is a function $\Phi : \mathcal{X} \mapsto \mathcal{X}$ such that

$$\Phi(\mathbf{x})_i = \sum_{j=1}^{n} w(\mathbf{x}_i, \mathbf{x}_j) v(\mathbf{x}_j) = \mathbf{W}(\mathbf{x})_i^\top \mathbf{V}(\mathbf{x}),$$

where $\mathbf{W}(\mathbf{x})_i$ denotes the i-th column of $\mathbf{W}(\mathbf{x})$, a weighting function, and $\mathbf{V}(\mathbf{x}) = [v(\mathbf{x}_1), \dots, v(\mathbf{x}_n)]^\top$, an embedding. In contrast to operations on local neighborhood such as convolutions, non-local operations theoretically account for long range dependencies between elements in the set. In attention and the context of neural networks, w is a *learned* function reflecting the *relevance* of each other elements $\mathbf{x}_j$ with respect to the element $\mathbf{x}_i$ being embedded and given the task at hand. In the context of the paper, we compare to a type of attention coined as *dot-product self-attention*, which can typically be found in the encoder part of the transformer architecture [27]. Transformers are neural network models relying mostly on a succession of an attention layer followed by a fully-connected layer. Transformers can be used in sequence-to-sequence tasks — in this setting, they have an encoder with self-attention and a decoder part with a variant of self-attention —, or in sequence to label tasks, with only the encoder part. The paper deals with the latter. The name self-attention means that the attention is computed using a dot-product of linear transformations of $\mathbf{x}_i$ and $\mathbf{x}_j$, and $\mathbf{x}$ attends to itself only. In its matrix formulation, dot-product self-attention is a non-local operation whose matching vector is

$$\mathbf{W}(\mathbf{x})_i = \text{Softmax}\left(\frac{W_Q \mathbf{x}_i \mathbf{x}^\top W_K^\top}{\sqrt{d_k}}\right),$$

where $W_Q \in \mathbb{R}^{n \times d_k}$ and $W_K \in \mathbb{R}^{n \times d_k}$ are learned by the network. In order to know which $\mathbf{x}_j$ are relevant to $\mathbf{x}_i$, the network computes scores between a query for $\mathbf{x}_i$ ($W_Q \mathbf{x}_i$) and keys of all the elements of $\mathbf{x}$ ($W_K \mathbf{x}$). The softmax turns the scores into a weight vector in the simplex. Moreover, a linear mapping $\mathbf{V}(\mathbf{x}) = W_V \mathbf{x}$, the values, is also learned. W_Q and W_K are often shared [14]. A drawback of such attention is that for a sequence of length n, the model has to store an attention matrix $\mathbf{W}$ with size $O(n^2)$. More details can be found in [27].

B Additional Details

B.1 Kernel Interpretation

Thanks to the gluing lemma [20], $\mathbf{P}_\mathbf{z}(\mathbf{x}, \mathbf{x}')$ is a valid transport plan and, empirically, a rough approximation of $\mathbf{P}(\mathbf{x}, \mathbf{x}')$. $K_\mathbf{z}$ can therefore be seen as a surrogate of a well-known kernel [23], defined as

$$K_{\text{OT}}(\mathbf{x}, \mathbf{x}') := \sum_{i,i'=1}^{n} \mathbf{P}(\mathbf{x}, \mathbf{x}')_{ii'} \kappa(\mathbf{x}_i, \mathbf{x}'_{i'}). \tag{5}$$

When the entropic regularization ε is equal to 0, K_{OT} is equivalent to the 2-Wasserstein distance $W_2(\mathbf{x}, \mathbf{x}')$ with ground metric d_κ induced by kernel κ. K_{OT} is generally not positive definite (see [20], Chapter 8.3) and computationally costly (the number of transport plans to compute is quadratic in the number of sets to process whereas it is linear for $K_\mathbf{z}$). Now, we show the relationship between this kernel and our kernel $K_\mathbf{z}$, which is proved in Appendix C.1.

Lemma B.1 (Relation between $\mathbf{P}(\mathbf{x}, \mathbf{x}')$ and $\mathbf{P}_\mathbf{z}(\mathbf{x}, \mathbf{x}')$ when $\varepsilon = 0$)**.** For any $\mathbf{x}$, $\mathbf{x}'$ and $\mathbf{z}$ in $\mathcal{X}$ with lengths n, n' and p, by denoting $W_2^\mathbf{z}(\mathbf{x}, \mathbf{x}') := \langle \mathbf{P}_\mathbf{z}(\mathbf{x}, \mathbf{x}'), d_\kappa^2(\mathbf{x}, \mathbf{x}') \rangle^{1/2}$ we have

$$|W_2(\mathbf{x}, \mathbf{x}') - W_2^\mathbf{z}(\mathbf{x}, \mathbf{x}')| \leq 2 \min(W_2(\mathbf{x}, \mathbf{z}), W_2(\mathbf{x}', \mathbf{z})). \tag{6}$$

This lemma shows that the distance $W_2^\mathbf{z}$ resulting from $K_\mathbf{z}$ is related to the Wasserstein distance W_2; yet, this relation should not be interpreted as an approximation error as our goal is not to approximate W_2, but rather to derive a trainable embedding $\Phi_\mathbf{z}(\mathbf{z})$ with good computational properties. Lemma B.1 roots our features and to some extent self-attention in a rich optimal transport literature.

B.2 Variant of Unsupervised Learning of Parameter $\mathbf{z}$

In the fashion of the Nyström approximation, the p elements of $\mathbf{z}$ can be defined as the centroids obtained by K-means applied to all features from training sets in $\mathcal{X}$. A corollary of Lemma B.1 suggests another algorithm: a bound on the deviation term between W_2 and $W_2^\mathbf{z}$ for m samples $(\mathbf{x}^1, \dots, \mathbf{x}^m)$ is indeed

$$\mathcal{E}^2 := \frac{1}{m^2} \sum_{i,j=1}^{m} |W_2(\mathbf{x}^i, \mathbf{x}^j) - W_2^\mathbf{z}(\mathbf{x}^i, \mathbf{x}^j)|^2 \leq \frac{4}{m} \sum_{i=1}^{m} W_2^2(\mathbf{x}^i, \mathbf{z}). \tag{7}$$

The right-hand term corresponds to the objective of the Wasserstein barycenter problem [8], which yields the mean of a set of empirical measures (here the $\mathbf{x}$'s) under the OT metric. The Wasserstein barycenter is therefore an attractive candidate for choosing $\mathbf{z}$. K-means can be seen as a particular case of Wasserstein barycenter when $m = 1$ [8, 12] and is faster to compute. In practice, both methods yield similar results, see Appendix D, and we thus chose K-means to learn $\mathbf{z}$ in unsupervised settings throughout the experiments.

B.3 Extensions

Integrating positional information into the embedding. The discussed embedding and kernel do not take the position of the features into account, which may be problematic when dealing with structured data such as images or sentences. To this end, we borrow the idea of convolutional kernel networks, or CKN [17, 18], and penalize the similarities exponentially with the positional distance between a pair of elements in the input and reference sequences. More precisely, we multiply $\mathbf{P}(\psi(\mathbf{x}), \mathbf{z})$ element-wise by a distance matrix $\mathbf{S}$ defined as:

$$\mathbf{S}_{ij} = e^{-\frac{1}{\sigma_{\text{pos}}^2}(i/n - j/p)^2},$$

and replace it in the embedding. With similarity weights based *both* on content and position, the kernel associated to our embedding can be viewed as a generalization of the CKNs (whose similarity weights are based on position only), with feature alignment based on optimal transport. When dealing with multi-dimensional objects such as images, we just replace the index scalar i with an index vector of the same spatial dimension as the object, representing the positions of each dimension.

Using multiple references. A naive reconstruction using different references $\mathbf{z}^1, \dots, \mathbf{z}^q$ in $\mathcal{X}$ may yield a better approximation of the transport plan. In this case, the embedding of $\mathbf{x}$ becomes

$$\Phi_{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}) = {}^1\!/\!_{\sqrt{q}}\left(\Phi_{\mathbf{z}^1}(\mathbf{x}), \dots, \Phi_{\mathbf{z}^q}(\mathbf{x})\right), \tag{8}$$

with q the number of references (the factor ${}^1\!/\!_{\sqrt{q}}$ comes from the mean). The references do not necessarily have the same number of elements $\mathbf{z}_i$. Using Eq. (6), we can obtain a deviation bound similar to (9) for a data set of m samples $(\mathbf{x}^1, \dots, \mathbf{x}^m)$ and q references (see Appendix C.2 for details). To choose multiple references, we tried a K-means algorithm with 2-Wasserstein distance for assigning clusters, and we updated the centroids as in the single-reference case. Using multiple references appears to be useful when optimizing $\mathbf{z}$ with supervision (see Appendix D).

B.4 On the Relationship between our Embedding and Self-attention

Our embedding and a single layer of transformer encoder, recalled in Appendix A, share the same type of inductive bias, *i.e*, aggregating features relying on similarity weights. We now clarify their relationship. Our embedding is arguably simpler (see respectively size of attention and number of parameters in Table 2), and may compete in some settings with the transformer self-attention as illustrated in Section 3.

Shared reference versus self-attention. There is a correspondence between the values, attention matrix in the transformer and φ, $\mathbf{P}$ in Definition 2.1, yet also noticeable differences. On the one hand, $\Phi_{\mathbf{z}}$ aligns a given sequence $\mathbf{x}$ with respect to a reference $\mathbf{z}$, learned with or without supervision, and shared across the data set. Our weights are computed using optimal transport. On the other hand, a transformer encoder performs self-alignment: for a given $\mathbf{x}_i$, features are aggregated depending on a similarity score between $\mathbf{x}_i$ and the elements of $\mathbf{x}$ only. The similarity score is a matrix product between queries Q and keys K matrices, learned with supervision and shared across the data set. In this regard, our work complements a recent line of research questioning the dot-product, learned self-attention [21, 31]. Self-attention-like weights can also be obtained with our embedding by computing $\mathbf{P}(\mathbf{x}, \mathbf{z}_i)\mathbf{P}(\mathbf{x}, \mathbf{z}_i)^\top$ for each reference i.

Position smoothing and relative positional encoding. Transformers can add an absolute positional encoding to the input features [27]. Yet, relative positional encoding [9] is a current standard for integrating positional information: the position offset between the query element and a given key can be injected in the attention score [26], which is equivalent to our approach. The link between CKNs and our kernel, provided by this positional encoding, stands in line with recent works casting

Table 2: Relationship between $\Phi_{\mathbf{z}}$ and transformer self-attention. k: a function describing how the transformer integrates positional information; n: sequence length; q: number of references or attention heads; d: dimension of the embeddings; p: number of supports in $\mathbf{z}$. Typically, $p \ll d$. In recent transformer architectures, positional encoding requires learning additional parameters ($\sim qd^2$).

	Self-Attention	$\Phi_{\mathbf{z}}$
Attention score	$\mathbf{W} = W^\top Q$	$\mathbf{P}$
Size of score	$O(n^2)$	$O(np)$
Alignment w.r.t:	$\mathbf{x}$ itself	$\mathbf{z}$
Learned + Shared	W and Q	$\mathbf{z}$
Nonlinear mapping	Feed-forward	φ or ψ
Position encoding	$k(t_i, t'_j)$	$e^{-\frac{1}{\sigma^2_{\text{pos}}}(\frac{i}{n} - \frac{j}{n'})^2}$
Nb. parameters	$\sim qd^2$	qpd
Supervision	Needed	Not needed

attention and convolution into a unified framework [1]. In particular, [6] show that attention learns convolution in the setting of image classification: the kernel pattern is learned at the same time as the filters.

Multiple references and attention heads. In the transformer architecture, the succession of blocks composed of an attention layer followed by a fully-connected layer is called a head, with each head potentially focusing on different parts of the input. Successful architectures have a few heads in parallel. The outputs of the heads are then aggregated to output a final embedding. A layer of our embedding with non-linear kernel κ can be seen as such a block, with the references playing the role of the heads. As some recent works question the role of attention heads [19, 28], exploring the content of our learned references $\mathbf{z}$ may provide another perspective on this question.

C Proofs

C.1 Proof of Lemma B.1

Proof. First, since $\sum_{j=1}^{n'} p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk} = 1$ for any k, we have

$$\begin{aligned} W_2(\mathbf{x}, \mathbf{z})^2 &= \sum_{i=1}^{n}\sum_{k=1}^{p} \mathbf{P}(\mathbf{x}, \mathbf{z})_{ik} d_\kappa^2(\mathbf{x}_i, \mathbf{z}_k) \\ &= \sum_{i=1}^{n}\sum_{k=1}^{p}\sum_{j=1}^{n'} p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik} d_\kappa^2(\mathbf{x}_i, \mathbf{z}_k) \\ &= \|\mathbf{u}\|_2^2, \end{aligned}$$

with $\mathbf{u}$ a vector in $\mathbb{R}^{nn'p}$ whose entries are $\sqrt{p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik}}\, d_\kappa(\mathbf{x}_i, \mathbf{z}_k)$ for $i = 1, \dots, n$, $j = 1, \dots, n'$ and $k = 1, \dots, p$. We can also rewrite $W_2^{\mathbf{z}}(\mathbf{x}, \mathbf{x}')$ as an ℓ_2-norm of a vector $\mathbf{v}$ in $\mathbb{R}^{nn'p}$ whose entries are $\sqrt{p\mathbf{P}(\mathbf{x}', \mathbf{z})_{jk}\mathbf{P}(\mathbf{x}, \mathbf{z})_{ik}}\, d_\kappa(\mathbf{x}_i, \mathbf{x}'_j)$. Then by Minkowski inequality for the

ℓ_2-norm, we have

$$\begin{aligned}
|W_2(\mathbf{x},\mathbf{z}) - W_2^{\mathbf{z}}(\mathbf{x},\mathbf{x}')| &= |\|\mathbf{u}\|_2 - \|\mathbf{v}\|_2| \\
&\leq \|\mathbf{u}-\mathbf{v}\|_2 \\
&= \left(\sum_{i=1}^{n}\sum_{k=1}^{p}\sum_{j=1}^{n'} p\mathbf{P}(\mathbf{x}',\mathbf{z})_{jk}\mathbf{P}(\mathbf{x},\mathbf{z})_{ik}(d_\kappa(\mathbf{x}_i,\mathbf{z}_k) - d_\kappa(\mathbf{x}_i,{\mathbf{x}'}_j))^2\right)^{1/2} \\
&\leq \left(\sum_{i=1}^{n}\sum_{k=1}^{p}\sum_{j=1}^{n'} p\mathbf{P}(\mathbf{x}',\mathbf{z})_{jk}\mathbf{P}(\mathbf{x},\mathbf{z})_{ik}d_\kappa^2({\mathbf{x}'}_j,\mathbf{z}_k)\right)^{1/2} \\
&= \left(\sum_{k=1}^{p}\sum_{j=1}^{n'} \mathbf{P}(\mathbf{x}',\mathbf{z})_{jk}d_\kappa^2({\mathbf{x}'}_j,\mathbf{z}_k)\right)^{1/2} \\
&= W_2(\mathbf{x}',\mathbf{z}),
\end{aligned}$$

where the second inequality is the triangle inequality for the distance d_κ. Finally, we have

$$\begin{aligned}
&|W_2(\mathbf{x},\mathbf{x}') - W_2^{\mathbf{z}}(\mathbf{x},\mathbf{x}')| \\
\leq &|W_2(\mathbf{x},\mathbf{x}') - W_2(\mathbf{x},\mathbf{z})| + |W_2(\mathbf{x},\mathbf{z}) - W_2^{\mathbf{z}}(\mathbf{x},\mathbf{x}')| \\
\leq &W_2(\mathbf{x}',\mathbf{z}) + W_2(\mathbf{x}',\mathbf{z}) \\
= &2W_2(\mathbf{x}',\mathbf{z}),
\end{aligned}$$

where the second inequality is the triangle inequality for the 2-Wasserstein distance. By symmetry, we also have $|W_2(\mathbf{x},\mathbf{x}') - W_2^{\mathbf{z}}(\mathbf{x},\mathbf{x}')| \leq 2W_2(\mathbf{x},\mathbf{z})$, which concludes the proof. □

A corollary is

$$\mathcal{E}^2 := \frac{1}{m^2}\sum_{i,j=1}^{m} |W_2(\mathbf{x}^i,\mathbf{x}^j) - W_2^{\mathbf{z}}(\mathbf{x}^i,\mathbf{x}^j)|^2 \leq \frac{4}{m}\sum_{i=1}^{m} W_2^2(\mathbf{x}^i,\mathbf{z}). \tag{9}$$

C.2 Relationship between W_2 and $W_2^{\mathbf{z}}$ for multiple references

Using the relation prooved in Appendix C.1, we can obtain a bound on the error term between W_2 and $W_2^{\mathbf{z}}$ for a data set of m samples $(\mathbf{x}^1,\dots,\mathbf{x}^m)$ and q references $(\mathbf{z}^1,\dots,\mathbf{z}^q)$

$$\mathcal{E}^2 := \frac{1}{m^2}\sum_{i,j=1}^{m} |W_2(\mathbf{x}^i,\mathbf{x}^j) - W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}^i,\mathbf{x}^j)|^2 \leq \frac{4}{mq}\sum_{i=1}^{m}\sum_{j=1}^{q} W_2^2(\mathbf{x}^i,\mathbf{z}^j). \tag{10}$$

When $q=1$, the right-hand term in the inequality is the objective to minimize in the Wasserstein barycenter problem [8], which further explains why we considered it: Once $W_2^{\mathbf{z}}$ is close to the Wasserstein distance W_2, $K_{\mathbf{z}}$ will also be close to K_{OT}. We extend here the bound in equation 9 in the case of one reference to the multiple-reference case. The approximate 2-Wasserstein distance $W_2^{\mathbf{z}}(\mathbf{x},\mathbf{x}')$ thus becomes

$$W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x},\mathbf{x}') := \left\langle \frac{1}{q}\sum_{j=1}^{q}\mathbf{P}_{\mathbf{z}^j}(\mathbf{x},\mathbf{x}'), d_\kappa^2(\mathbf{x},\mathbf{x}')\right\rangle^{1/2} = \left(\frac{1}{q}\sum_{j=1}^{q} W_2^{\mathbf{z}^j}(\mathbf{x},\mathbf{x}')^2\right)^{1/2}.$$

Then by Minkowski inequality for the ℓ_2-norm we have

$$\begin{aligned}
|W_2(\mathbf{x},\mathbf{x}') - W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x},\mathbf{x}')| &= \left|\left(\frac{1}{q}\sum_{j=1}^{q} W_2(\mathbf{x},\mathbf{x}')^2\right)^{1/2} - \left(\frac{1}{q}\sum_{j=1}^{q} W_2^{\mathbf{z}^j}(\mathbf{x},\mathbf{x}')^2\right)^{1/2}\right| \\
&\leq \left(\frac{1}{q}\sum_{j=1}^{q}(W_2(\mathbf{x},\mathbf{x}') - W_2^{\mathbf{z}^j}(\mathbf{x},\mathbf{x}'))^2\right)^{1/2},
\end{aligned}$$

and by equation 9 we have

$$|W_2(\mathbf{x},\mathbf{x}') - W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x},\mathbf{x}')| \leq \left(\frac{4}{q}\sum_{j=1}^{q}\min(W_2(\mathbf{x},\mathbf{z}^j), W_2(\mathbf{x}',\mathbf{z}^j))^2\right)^{1/2}.$$

Finally the approximation error in terms of Frobenius is bounded by

$$\mathcal{E}^2 := \frac{1}{m^2}\sum_{i,j=1}^{m}|W_2(\mathbf{x}^i,\mathbf{x}^j) - W_2^{\mathbf{z}^1,\dots,\mathbf{z}^q}(\mathbf{x}^i,\mathbf{x}^j)|^2 \leq \frac{4}{mq}\sum_{i=1}^{m}\sum_{j=1}^{q}W_2^2(\mathbf{x}^i,\mathbf{z}^j).$$

In particular, when $q = 1$ that is the case of single reference, we have

$$\mathcal{E}^2 \leq \frac{4}{m}\sum_{i=1}^{m}W_2^2(\mathbf{x}^i,\mathbf{z}),$$

where the right term equals to the objective of the Wasserstein barycenter problem, which justifies the choice of $\mathbf{z}$ when learning without supervision.

D Additional Experiments and Setup Details

This section contains additional experiments on CIFAR-10, whose purpose is to illustrate the kernel associated with our embedding with respect to other classical or optimal transport based kernels, and test our embedding on another data modality; additional results for SCOP 1.75, state-of-the-art results for detection of chromatin profiles and promising results in NLP; details on our setup, in particular hyper-parameter tuning for our methods and the baselines.

D.1 Experiments on Kernel Matrices (only for small data sets).

Here, we compare the optimal transport kernel K_{OT} (5) and its surrogate $K_{\mathbf{z}}$ (2) (with $\mathbf{z}$ learned without supervision) to common and other OT kernels. Although our embedding $\Phi_{\mathbf{z}}$ is scalable, the exact kernel require the computation of Gram matrices. For this toy experiment, we therefore use 5000 samples only of CIFAR-10 (images with 32×32 pixels), encoded without supervision using a two-layer convolutional kernel network [17]. The resulting features are 3×3 patches living in $\mathbb{R}^d$ with $d = 256$ or 8192. K_{OT} and $K_{\mathbf{z}}$ aggregate existing features depending on the ground cost defined by $-\kappa$ (Gaussian kernel) given the computed weight matrix $\mathbf{P}$. In that sense, we can say that these kernels work as an adaptive pooling. We therefore compare it to kernel matrices corresponding to mean pooling and no pooling at all (linear kernel). We also compare to a recent positive definite and fast optimal transport based kernel, the Sliced Wasserstein Kernel [15] with 10, 100 and 1000 projection directions. We add a positional encoding to it so as to have a fair comparison with our kernels. A linear classifier is trained from this matrices. Although we cannot prove that K_{OT} is positive definite, the classifier trained on the kernel matrix converges when ε is not too small. The results can be seen on Table 3. Without positional information, our kernels do better than Mean pooling. When the positions are encoded, the Linear kernel is also outperformed. Note that including positions in Mean pooling and Linear kernel means interpolating between these two kernels: in the Linear kernel, only patches with same index are compared while in the Mean pooling, all patches are compared. All interpolations did worse than the Linear kernel. The runtimes illustrate the scalability of $K_{\mathbf{z}}$.

D.2 CIFAR-10

Here, we test our embedding on the same data modality: we use CIFAR-10 features, *i.e.*, $60,000$ images with 32×32 pixels and 10 classes encoded using a two-layer CKN [17], one of the baseline architectures for unsupervised learning of CIFAR-10, and evaluate on the standard test set. The very best configuration of the CKN yields a small number (3×3) of high-dimensional ($16,384$) patches and an accuracy of 85.8%. We will illustrate our embedding on a configuration which performs slightly less but provides more patches (16×16), a setting for which it was designed.

The input of our embedding are unsupervised features extracted from a 2-layer CKN with kernel sizes equal to 3 and 3, and Gaussian pooling size equal to 2 and 1. We consider the following

Table 3: Classification accuracies for 5000 samples of CIFAR-10 using CKN features [17] and forming Gram matrix. A random baseline would yield 10%.

Dataset	(3×3), 256	
Kernel	Accuracy	Runtime
Mean Pooling	58.5	$\sim$ 30 sec
Flatten	67.6	$\sim$ 30 sec
Sliced-Wasserstein [15]	63.8	$\sim$ 2 min
Sliced-Wasserstein [15] + sin. pos enc. [10]	66.0	$\sim$ 2 min
K_{OT}	64.5	$\sim$ 20 min
K_{OT} + our pos enc.	67.1	$\sim$ 20 min
$K_{\mathbf{z}}$	67.9	$\sim$ 30 sec
$K_{\mathbf{z}}$ + our pos enc.	70.2	$\sim$ 30 sec

Table 4: Hyperparameter search range for CIFAR-10

Hyperparameter	Search range
Entropic regularization ε	$[1.0; 0.1; 0.01; 0.001]$
Position encoding bandwidth σ_{pos}	$[0.5; 0.6; 0.7; 0.8; 0.9; 1.0]$

configurations of the number of filters at each layer, to simulate two different input dimensions for our embedding:

- 64 filters at first and 256 at second layer, which yields a $16 \times 16 \times 256$ representation for each image.
- 256 filters at first and 1024 at second layer, which yields a $16 \times 16 \times 1024$ representation for each image.

Since the features are the output of a Gaussian embedding, κ in our embedding will be the linear kernel. The embedding is learned with one reference and various supports using K-means method described in Section 2, and compared to several classical pooling baselines, including the original CKN's Gaussian pooling with pooling size equal to 6. The hyper-parameters are the entropic regularization ε and bandwidth for position encoding σ_{pos}. Their search grids are shown in Table 4 and the results in Table 5. Without supervision, the adaptive pooling of the CKN features by our embedding notably improves their performance. We notice that the position encoding is very important to this task, which substantially improves the performance of its counterpart without it.

Table 5: Classification results using unsupervised representations for CIFAR-10 for two feature configurations (extracted from a 2-layer unsipervised CKN with different number of filters). We consider here our embedding with one reference and different number of supports, learned with K-means, with or without position encoding (PE).

Method	Nb. supports	$16 \times 16 \times 256$	$16 \times 16 \times 1024$
Flatten		73.1	76.1
Mean pooling		64.9	73.4
Gaussian pooling [17]		77.5	82.0
Ours	9	75.6	79.3
Ours (with PE)		78.0	82.2
Ours	64	77.9	80.1
Ours (with PE)		81.4	83.2
Ours	144	78.4	80.7
Ours (with PE)		81.8	83.4

Table 6: Hyperparameter search grid for SCOP 1.75

Hyperparameter	Search range
ε for Sinkhorn	$[1.0; 0.5; 0.1; 0.05; 0.01]$
λ for classifier (unsupervised setting)	$1/2^{\text{range}(5,20)}$
λ for classifier (supervised setting)	[1e-6;1e-5;1e-4;1e-3]

Table 7: Hyperparameter search grid for SCOP 1.75 baselines.

Model and Hyperparameter	Search range
ApproxRepSet: Hidden Sets $\times$ Cardinality	$[20; 30; 50; 100] \times [10; 20; 50]$
ApproxRepSet: Learning Rate	$[0.0001; 0.0005; 0.001]$
ApproxRepSet: Weight Decay	[1e-5;1e-4;1e-3;1e-2]
Set Transformer: Heads $\times$ Dim Hidden	$[1; 4; 8] \times [64; 128; 256]$
Set Transformer: Learning Rate	$[0.0001; 0.0005; 0.001]$
Set Transformer: Weight Decay	[1e-5;1e-4;1e-3;1e-2]

D.3 Protein fold recognition

Dataset description. Our protein fold recognition experiments consider the Structural Classification Of Proteins (SCOP) version 1.75 and 2.06. We follow the data preprocessing protocols in [13], which yields a training and validation set composed of 14699 and 2013 sequences from SCOP 1.75, and a test set of 2533 sequences from SCOP 2.06. The resulting protein sequences belong to 1195 different folds, thus the problem is formulated as a multi-classification task. The input sequence is represented as a 45-dimensional vector at each amino acid. The vector consists of a 20-dimensional one-hot encoding of the sequence, a 20-dimensional position-specific scoring matrix (PSSM) representing the profile of amino acids, a 3-class secondary structure represented by a one-hot vector and a 2-class solvent accessibility. The lengths of the sequences are varying from tens to thousands.

Models setting and hyperparameters. We consider here the one-layer models followed by a global mean pooling for the baseline methods CKN [4] and RKN [5]. We build our model on top of the one-layer CKN model, where κ can be seen as a Gaussian kernel on the k-mers in sequences. The only difference between our model and CKN is thus the pooling operation, which is given by our embedding introduced in Section 2. The bandwidth parameter of the Gaussian kernel κ on k-mers is fixed to 0.6 for unsupervised models and 0.5 for supervised models, the same as used in CKN which were selected by the accuracy on the validation set. The filter size k is fixed to 10 and different numbers of anchor points in Nyström for κ are considered in the experiments. The other hyperparameters for our embedding are the entropic regularization parameter ε, the number of supports in a reference p, the number of references q, the number of iterations for Sinkhorn's algorithm and the regularization parameter λ in the linear classifier. The search grid for ε and λ is shown in Table 6 and they are selected by the accuracy on validation set. ε plays an important role in the performance and is observed to be stable for the same dataset. For this dataset, it is selected to be 0.5 for all the unsupervised and supervised models. The effect of other hyperparameters will be discussed below.

For the baseline methods, the accuracies of PSI-BLAST and DeepSF are taken from [13]. The hyperparameters for CKN and RKN can be found in [5]. For Rep the Set [25] and Set Transformer [16], we use the public implementations by the authors. These two models are used on the top of a convolutional layer of the same filter size as CKN to extract k-mer features. As the exact version of Rep the Set does not provide any implementation for back-propagation to a bottom layer of it, we consider the approximate version of Rep the Set only, which also scales better to our dataset. The default architecture of Set Transformer did not perform well due to overfitting. We therefore used a shallower architecture with one ISAB, one PMA and one linear layer, similar to the one-layer architectures of CKN and our model. We tuned their model hyperparameters, weight decay and learning rate. The search grids for these hyperparameters are shown in Table 7.

Unsupervised embedding. The kernel embedding φ, which is infinite dimensional for the Gaussian kernel, is approximated with the Nyström method using K-means on 300000 k-mers extracted

Table 8: Classification accuracy (top 1/5/10) results of our unsupervised embedding for SCOP 1.75. We show the results for different combinations of (number of references q $\times$ number of supports p). The reference measures $\mathbf{z}$ are learned with either K-means or Wasserstein barycenter for updating centroids.

Nb. filters	Method	q	Embedding size ($q \times p$)			
			10	50	100	200
128	K-means	1	76.5/91.5/94.4	77.5/91.7/94.5	79.4/92.4/94.9	78.7/92.1/95.1
		5	72.8/89.9/93.7	77.8/91.7/94.6	78.6/91.9/94.6	78.1/92.1/94.7
		10	62.7/85.8/91.1	76.5/91.0/94.2	78.1/92.2/94.9	78.6/92.2/94.7
	Wass. bary.	1	64.0/85.9/91.5	71.6/88.9/93.2	77.2/91.4/94.2	77.5/91.9/94.8
		5	70.5/89.1/93.0	76.6/91.3/94.4	78.4/91.7/94.3	77.1/91.9/94.7
		10	63.0/85.7/91.0	75.9/91.4/94.3	77.5/91.9/94.6	77.7/92.0/94.7
1024	K-means	1	84.4/95.0/96.6	84.6/95.0/97.0	85.7/95.3/96.7	85.4/95.2/96.7
		5	81.1/94.0/96.2	84.9/94.8/96.8	84.7/94.4/96.7	85.2/95.0/96.7
		10	79.8/93.5/95.9	83.1/94.6/96.6	84.4/94.7/96.7	84.8/94.9/96.7

Table 9: Classification accuracy (top 1/5/10) of supervised models for SCOP 1.75. The accuracies obtained by averaging 10 different runs. We show the results of using either one reference with 50 supports or 5 references with 10 supports. Here DeepSF is a 10-layer CNN model.

Method	Runtime	Top 1/5/10 accuracy on SCOP 2.06	
PSI-BLAST [13]	-	84.53/86.48/87.34	
DeepSF [13]	-	73.00/90.25/94.51	
Set Transformer [16]	3.3h	79.15±4.61/91.54±1.40/94.33±0.63	
ApproxRepSet [25]	2h	84.51±0.58/94.03±0.44/95.73±0.37	
Number of filters		128	512
CKN [4]	1.5h	76.30±0.70/92.17±0.16/95.27±0.17	84.11±0.11/94.29±0.20/96.36±0.13
RKN [5]	-	77.82±0.35/92.89±0.19/95.51±0.20	85.29±0.27/94.95±0.15/96.54±0.12
Ours			
$\Phi_{\mathbf{z}}$ (1×50)	3.5h	82.83±0.41/93.89±0.33/96.23±0.12	88.40±0.22/95.76±0.13/97.10±0.15
$\Phi_{\mathbf{z}}$ (5×10)	4h	**84.68±0.50/94.68±0.29/96.49±0.18**	**88.66±0.25/95.90±0.15/97.33±0.14**

from the same training set as in [5]. The reference measures are learned by using either K-means or Wasserstein to update centroids in 2-Wasserstein K-means on 3000 subsampled sequences for RAM-saving reason. We evaluate our model on top of features extracted from CKNs of different dimensions, representing the number of anchor points used to approximate κ. The number of iterations for Sinkhorn is fixed to 100 to ensure the convergence. The results for different combinations of q and p are provided in Table 8. Increasing the number of supports p can improve the performance and then saturate it when p is too large. On the other hand, increasing the number of references while keeping the embedding dimension (*i.e.* $p \times q$) constant is not significantly helpful in this unsupervised setting. We also notice that Wasserstein Barycenter for learning the references does not outperform K-means, while the latter is faster in terms of computation.

Supervised embedding. Our supervised embedding is initialized with the unsupervised method and then trained in an alternating fashion which was also used for CKN: we use an Adam algorithm to update anchor points in Nyström and reference measures $\mathbf{z}$, and the L-BFGS algorithm to optimize the classifier. The learning rate for Adam is initialized with 0.01 and halved as long as there is no decrease of the validation loss for 5 successive epochs. In practice, we notice that using a small number of Sinkhorn iterations can achieve similar performance to a large number of iteration, while being much faster to compute. We thus fix it to 10 throughout the experiments. The accuracy results are obtained by averaging on 10 runs with different seeds following the setting in [5]. The results are shown in Table 9 with error bars. The effect of the number of supports q is similar to the unsupervised case, while increasing the number of references can indeed improve performance.

Table 10: Model architecture for DeepSEA dataset.

Model architecture
Conv1d(in channels=4, out channels=d, kernel size=16) + ReLU
(Ours) EmbeddingLayer(in channels=d, supports=64, references=1, $\varepsilon = 1.0$, PE=True, $\sigma_{\text{pos}} = 0.1$)
Linear(in channels=d, out channels=d) + ReLU
Dropout(0.4)
Linear(in channels=$d \times 64$, out channels=919) + ReLU
Linear(in channels=919, out channels=919)

Table 11: Results for prediction of chromatin profiles on the DeepSEA dataset. The metrics are area under ROC (auROC) and area under PR curve (auPRC), averaged over 919 chromatin profiles. The accuracies are averaged from 10 different runs. Armed with the positional encoding (PE) described in Section 2, our embedding outperforms the state-of-the-art model and another model of our embedding with the PE proposed in [27].

Method	DeepSEA	Ours	Ours ($d = 1024$)	Ours ($d = 1536$)
Position encoding	-	Sinusoidal [27]	Ours	Ours
auROC	0.933	0.917	0.935	**0.936**
auPRC	0.342	0.311	0.354	**0.360**

D.4 Detection of chromatin profiles

Dataset description. Predicting the functional effects of noncoding variants from only genomic sequences is a central task in human genetics. A fundamental step for this task is to simultaneously predict large-scale chromatin features from DNA sequences [34]. We consider here the DeepSEA dataset, which consists in simultaneously predicting 919 chromatin profiles including 690 transcription factor (TF) binding profiles for 160 different TFs, 125 DNase I sensitivity profiles and 104 histone-mark profiles. In total, there are 4.4 million, 8000 and 455024 samples for training, validation and test. Each sample consists of a 1000-bp DNA sequence from the human GRCh37 reference. Each sequence is represented as a 1000×4 binary matrix using one-hot encoding on DNA characters. The dataset is available at `http://deepsea.princeton.edu/media/code/deepsea_train_bundle.v0.9.tar.gz`. Note that the labels for each profile are very imbalanced in this task with few positive samples. For this reason, learning unsupervised models could be intractable as they may require an extremely large number of parameters if junk or redundant sequences cannot be filtered out.

Model architecture and hyperparameters. For the above reason and fair comparison, we use here our supervised embedding as a module in Deep NNs. The architecture of our model is shown in Table 10. We use an Adam optimizer with initial learning rate equal to 0.01 and halved at epoch 1, 4, 8 for 15 epochs in total. The number of iterations for Sinkhorn is fixed to 30. The whole training process takes about 30 hours on a single GTX2080TI GPU. The dropout rate is selected to be 0.4 from the grid $[0.1; 0.2; 0.3; 0.4; 0.5]$ and the weight decay is 1e-06, the same as [34]. The σ_{pos} for position encoding is selected to be 0.1, by the validation accuracy on the grid $[0.05; 0.1; 0.2; 0.3; 0.4; 0.5]$. The checkpoint with the best validation accuracy is used to evaluate on the test set. Area under ROC (auROC) and area under precision curve (auPRC), averaged over 919 chromatin profiles, are used to measure the performance. The hidden size d is chosen to be either 1024 or 1536.

Results and importance of position encoding. We compare our model to the state-of-the-art CNN model DeepSEA [34] with 3 convolutional layers, whose best hyper-parameters can be found in the corresponding paper. Our model outperforms DeepSEA, while requiring fewer layers. The positional information is known to be important in this task. To show the efficacy of our position encoding, we compare it to the sinusoidal encoding used in the original transformer [27]. We observe that our encoding with properly tuned σ_{pos} requires fewer layers, while being interpretable from a kernel point of view. We also find that larger hidden size d performs better, as shown in Table 11. ROC and PR curves for all the chromatin profiles and stratified by transcription factors, DNase I-hypersensitive sites and histone-marks can also be found in Figure 2.

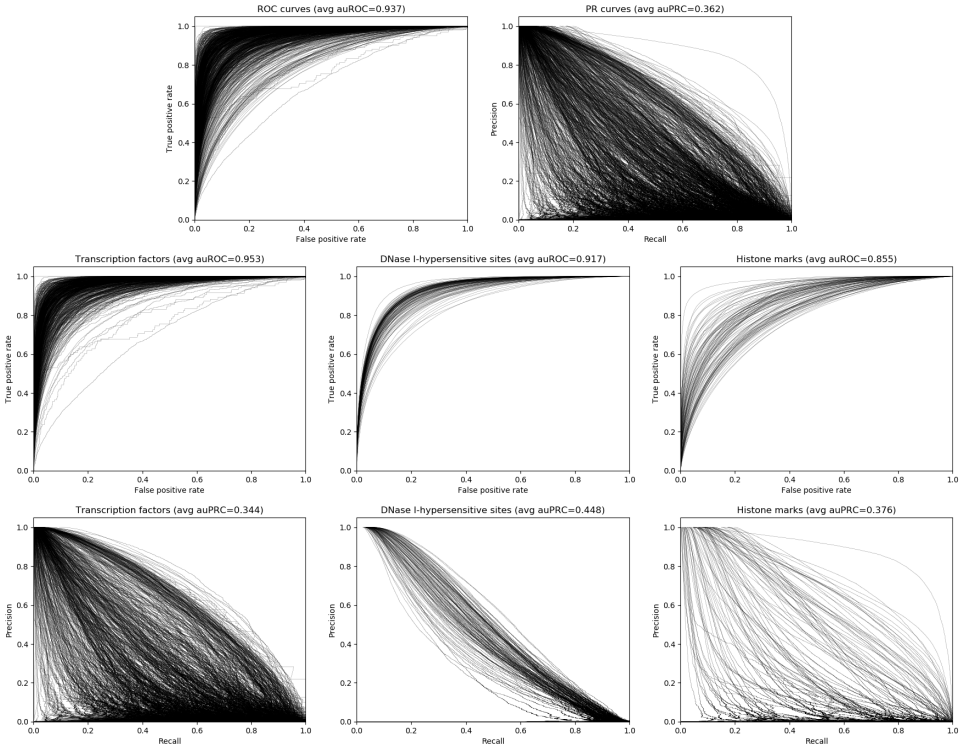

Figure 2: ROC and PR curves for all the chromatin profiles (first row) and stratified by transcription factors (left column), DNase I-hypersensitive sites (middle column) and histone-marks (right column). The profiles with positive samples fewer than 50 on the test set are not taken into account.

D.5 SST-2

Dataset description. The data set contains 67,349 training samples and 872 validation samples and can be found at `https://gluebenchmark.com/tasks`. The test set contains 1,821 samples for which the predictions need to be submitted on the GLUE leaderboard, with limited number of submissions. As a consequence, our training and validation set are extracted from the original training set (80% of the original training set is used for our training set and the remaining 20% is used for our validation set), and we report accuracies on the standard validation set, used as a test set. The reviews are padded with zeros when their length is shorter than the chosen sequence length (we choose 30 and 66, the latter being the maximum review length in the data set) and the BERT implementation requires to add special tokens [CLS] and [SEP] at the beginning and the end of each review.

Model architecture and hyperparameters. In most transformers such as BERT, the embedding associated to the token [CLS] is used for classification and can be seen in some sense as an embedding of the review adapted to the task. The features we used are the word features provided by the BERT base-uncased version, available at `https://huggingface.co/transformers/pretrained_models.html`. For this version, the dimension of the word features is 768. Our model is one layer of our embedding, with φ the Gaussian kernel mapping with varying number of Nyström filters in the supervised setting, and the Linear kernel in the unsupervised setting. We do not add positonnal encoding as it is already integrated in BERT features. In the unsupervised setting, the output features are used to train a large-scale linear classifier, a Pytorch linear layer. We choose the best hyper-parameters based on the accuracy of a validation set. In the supervised case, the parameters of the previous model, $\mathbf{w}$ and $\mathbf{z}$, are optimized end-to-end. In this case, we tune the learning rate. In both case, we tune the entropic regularization parameter of optimal transport and the bandwidth of the Gaussian kernel. The parameters in the search grid are summed up in Table 13. The best entropic regularization and Gaussian kernel bandwidth are typically and respectively 3.0 and 0.5 for this data set. The supervised training process takes between half an hour for smaller

Table 12: Accuracies on standard validation set for SST-2 with our unsupervised features depending on the number of references and supports. The references were computed using K-means on samples for multiple references and K-means on patches for multiple supports. The size of the input BERT features is (length $\times$ dimension). The accuracies are averaged from 10 different runs.

BERT Input Feature Size	(30×768)		(66×768)	
Features	Pre-trained	Fine-tuned	Pre-trained	Fine-tuned
[CLS]	84.6±0.3	90.3±0.1	86.0±0.2	**92.8±0.1**
Flatten	84.9±0.4	**91.0±0.1**	85.2±0.3	92.5±0.1
Mean pooling	85.3±0.3	90.8±0.1	85.4±0.3	92.6±0.2
$\Phi_{\mathbf{z}}$ (1×3)	85.5±0.1	90.9±0.1	86.5±0.1	92.6±0.1
$\Phi_{\mathbf{z}}$ (1×10)	85.1±0.4	90.9±0.1	85.9±0.3	92.6±0.1
$\Phi_{\mathbf{z}}$ (1×30)	86.3±0.3	90.8±0.1	86.6±0.5	92.6±0.1
$\Phi_{\mathbf{z}}$ (1×100)	85.7±0.7	90.9±0.1	86.6±0.1	92.7±0.1
$\Phi_{\mathbf{z}}$ (1×300)	**86.8±0.3**	90.9±0.1	**87.2±0.1**	92.7±0.1

Table 13: Hyperparameter search grid for SST-2.

Hyperparameter	Search range
Entropic regularization ε	$[10.0; 3.0; 1.0; 0.5]$
λ for classifier (unsupervised setting)	$10^{\text{range}(-10,1)}$
Gaussian kernel bandwidth	$[0.4; 0.4; 0.5; 0.6; 0.7; 0.8]$
Learning rate (supervised setting)	$[0.1; 0.01; 0.001]$

models (typically 128 filters in $\mathbf{w}$ and 3 supports in $\mathbf{z}$) and a few hours for larger models (256 filters and 100 supports) on a single GTX2080TI GPU. The hyper-parameters of the baselines were similarly tuned, see 14. Mean Pooling and [CLS] embedding did not require any tuning except for the regularization λ of the classifier, which followed the same grid as in Table 13.

Results and discussion. As explained in Section 3, our unsupervised embedding improves the BERT pre-trained features while still using a simple linear model as shown in Table 12, and its supervised counterpart enables to get even closer to the state-of-the art (for the BERT base-uncased model) accuracy, which is usually obtained after fine-tuning of the BERT model on the whole data set. This can be seen in Tables 15; 16. We also add a baseline consisting of one layer of classical self-attention, which did not do well hence was not reported in the main text.

Table 14: Hyperparameter search grid for SST-2 baselines.

Model and Hyperparameter	Search range
RepSet and ApproxRepSet: Hidden Sets $\times$ Cardinality	$[4; 20; 30; 50; 100] \times [3; 10; 20; 30; 50]$
ApproxRepSet: Learning Rate	$[0.0001; 0.001; 0.01]$
Set Transformer: Heads $\times$ Dim Hidden	$[1; 4] \times [8; 16; 64; 128]$
Set Transformer: Learning Rate	$[0.001; 0.01]$

Table 15: Classification accuracy on standard validation set of supervised models for SST-2, with pre-trained BERT (30×768) features. The accuracies of our embedding were averaged from 3 different runs before being run 10 times for the best results for comparison with baselines, cf. Section 3. 10 Sinkhorn iterations were used. We show the results of using either one reference with various supports or 4 references with various supports.

Method	Accuracy on SST-2		
Number of Nyström filters	32	64	128
$\Phi_{\mathbf{z}}$ (1×3)	88.38	88.38	88.18
$\Phi_{\mathbf{z}}$ (1×10)	88.11	88.15	87.61
$\Phi_{\mathbf{z}}$ (1×30)	88.30	88.30	88.26
$\Phi_{\mathbf{z}}$ (4×3)	88.07	88.26	88.30
$\Phi_{\mathbf{z}}$ (4×10)	87.6	87.84	88.11
$\Phi_{\mathbf{z}}$ (4×30)	88.18	$\mathbf{88.68}$	88.07

Table 16: Classification accuracy on standard validation set of all baselines for SST-2, with pre-trained BERT (30×768) features, averaged from 10 different runs.

Method	Accuracy on SST-2
[CLS] embedding [10]	90.3±0.1
Mean Pooling of BERT features [10]	$\mathbf{90.8 \pm 0.1}$
One Self-Attention Layer [27]	83.7±0.1
Approximate Rep the Set [25]	86.8±0.9
Rep the Set [25]	87.1±0.5
Set Transformer [16]	87.9±0.8
$\Phi_{\mathbf{z}}$ (1×30) (dot-product instead of OT)	86.9±1.1